R8C QuickDesign Guide

Introduction

The purpose of the document is to help the designed get started their design on the R8C/25 family. This document covers some commonly asked questions and provides some design tips. The document is not intended to be a replacement for the hardware manual; it is intended to supplement the manual by highlighting some key items most engineers will need to start their own design. It also discusses some design decisions from an application point of view

Contents

1. Power Supplies and Bypassing

1.1. Operating Frequency

The maximum operating frequency for the R8C/25 is dependent on the power supply voltage supplied to the R8C. The electrical characteristics section of the hardware manual defines the maximum frequency vs. voltage.

Input Voltage	Maximum Operating Frequency
3.0 VDC $\leq$ Vcc $\leq$ 5.5VDC	20 MHz
2.7 VDC $\leq$ Vcc < 3.0 VDC	10 MHz
2.2 VDC $\leq$ Vcc < 2.7 VDC	5 MHz

1.2 Bypassing

Bypass recommendations are listed in the usage notes and the
AD Converter section of the Hardware manual. The following are minimum bypass recommendations

- 0.1 uF ceramic from Vcc to Vss
- 0.1 uF ceramic from Vref to AVss (when using ADC)
-

All bypass capacitors should be connected using the shortest trace length possible and thick traces. Though not specifically required a 100 pF capacitor from analog input pins to AVss may help A/D converter noise rejection, this is discussed in more detail in the ADC section of this guide

2. Memory

Some R8C/25 devices have memory that exceeds the 64K address boundary. For application tips associated with utilizing this memory please see the Application Note Using R8C Expanded Memory.

3. Reset Requirements and the Reset Circuit

The R8C/25 has a Power-On-Reset (POR) circuit. This circuit provides a simple means to provide a stable reset signal to the MCU without the need for an external Reset IC. The POR circuit of the MCU should be sufficient for most designs; however, the POR only provides an internal Reset signal so there must be a system consideration if a System Reset signal is required. Under these conditions an MCU port pin can be used to provide that signal see Application Brief ********

3.1. Using the POR feature

Section 5.2 of the Hardware manual describes using the POR feature of the device. To use this feature a 5K (approximate value) resistor is connected between Vcc and the Reset pin. That is all that is required to use the POR feature.
To use the POR feature the Vcc must rise above 2.4V to release the Reset signal. The voltage detection circuit that is used for the POR asserts at 2.2V, which is the minimum operating voltage of the device, and does not release until the voltage rises above 2.4V. (See electrical characteristics Voltage Detection 0 and Power On Reset, Voltage Monitor 0 sections for full characteristics)

When using the POR feature do not connect a capacitor from the Reset input to Ground or you may have conditions where the POR circuit does not operate properly

3.2. Using RC Reset circuit

If the POR feature cannot be used, or is not desired, the MCU can be reliably Reset using an RC circuit connected to the Reset input. In this case the POR feature should be disabled by modifying the setting in the OFS register since the default setting enables the POR feature.
When using the RC circuit the time constant must keep the voltage on the Reset pin <0.2 X Vcc Min (0.44 volts) for a time equal to the power supply stabilization time + 20 clock cycles of the slow speed internal oscillator. The timing requirement starts after the Vcc has reached the minimum operating voltage. Table 20.12 list the power supply stabilization time td(P-R) as 2 mSec max.
Assuming that the ramp time on the Vcc is negligible the following calculation will provide values that can provide the required delay.
The calculation uses the following parameters for approximating values required

- 2.2 mSec time minimum delay for margin
- Step Vcc to 5.5 VDC lower step voltages require less time constant
- Negligible rise time for power supply. When power supply ramp times are greater than (xxx) the RC reset method can be difficult to apply

$$RC = -\,td(P\text{-}R) / \ln(1 - V_{reset}[max]/V_{cc})$$

$$-2.2\ \text{mSec} / \ln(1 - (.44V/5.5V))$$

$$RC \approx 27\ \text{mSec}$$

Though any resistor and capacitor values could be used that would provide the required delay a few guidelines should be considered:

1. Higher resistor values increase noise susceptibility concerns
2. For noise and stability ceramic capacitor values are desirable
3. Large values of capacitance will store higher energy levels which must be discharged to pull the reset line low

Based on these guidelines, typical values of resistance between 27k and 100k are reasonable values. These resistor values allow using readily available ceramic capacitors.
A diode is shown in parallel with the resistor in the Hardware manual. This diode provides the capacitor with a low resistance discharge path when the line is to be pulled low. Without this diode the capacitor will discharge through the internal parasitic diode of the Reset pin, this can cause problems if the energy stored in the capacitor is large.

3.3. Using External Reset circuit

If an external reset IC is desired and an on-chip debugger, for example the E8a, will be used the Reset IC should have an open-drain output so the debugger can control the Reset line. The Debugging section of this document provides additional information

3.4. MCU States after Hardware Reset

The state of each peripheral after Reset is given in the applicable sections of the HW manual. The Reset condition of each register is also given above the register description. Section 4 of the HW manual , Special Function Registers, provides a single table with all the peripheral register states after Reset.
A few key Reset conditions are listed below

- All GPIO are configured as inputs (High Impedance)
- The internal low-speed oscillator (fOCO-S) is the only clock running
- The CPU clock divider is set for divide by 8 (the CPU core is running at approximately 125 kHz/8)
- Watchdog Timer (WDT) is not running unless the automatic start feature is selected (see WDT section)
- App Brief on setting WDT

4. Voltage Detect

The R8C/25 has a three level voltage detection circuit which can be used for generating Interrupts or Reset signals. The voltage levels are fixed. The three setpoints and nominal trip points are listed below. The electrical characteristic section of the HW manual should be referenced for specific trip setpoints and conditions (Table 20.6 – 20.8)

Detection Level	Nominal Trip Voltage	Setpoint Action
Vdet0	2.3V±0.1	Reset
Vdet1	2.85 ±0.15V	Reset or Interrupt
Vdet2	3.6 ±0.3V	Reset or Interrupt

5. General Purpose I/O Circuit

The GPIO section of the HW manual describes exact pin configurations based on peripheral selection and other register settings. Some general information is listed below.

5.1. Setting Up and Using the GPIO

- Select a pin as an output by writing a “1” to the corresponding Data Direction Register (DDR)
- The Data Direction Register (DDR) is read/write. Setting the value to a 1 selects the pin as an output. Default state for GPIO is “0” (input)
- The Data Register for ports are read/write. When the Data Register is read with the port selected as an output the state of the output data latch (not the pin level) is read. When the port is configured as an input the pin level is read
- GPIO require CMOS level inputs (High ≥0.8 * Vcc, Low≤ 0.2*Vcc) see electrical characteristics for more information

- **Port 0 direction register is protected from accidental modification. Before modifying this register Bit 2 in the Protect Register (PRCR) must be Set. The protection bit is automatically Reset by the next internal write operation of the MCU; therefore the setting of the Protect bit and the modification of the direction register should be consecutive instructions with interrupts disabled during this time (see the Protection Section for more information)**
- When setting a pin as an output it is recommended that the desired output value of the port be written to the data latch first, then the direction register is set to an output. Though not important in all systems, this prevents an unintended output glitch on the port being setup.
- P4_2 can be used as an input if AD is not used. If AD is used this pin must have a Vref voltage applied to it (see ADC section for more information)

5.2. Internal Pull-Up Resistors

- Most GPIO can have internal pull-up resistors enabled when they are used as an input. The pullup resistors are typically applied as groups of pins, see the HW manual for the grouping.
- If any pin is set to an output by setting the associated DDR to 1 the internal pullup for that pin will be disabled.

5.3. Unassigned pins–

See section 7.5 of the HW manual for specific recommendations. For most GPIO the designer has two options with regard to handling unused pins. If a pin is just left floating it can consume extra power and is leaves the system more susceptible to noise problems.

The first option is to set the pin to an input (the default state after Reset) and connect the pin to Vcc or Vss using a resistor. There is no difference from an MCU standpoint to one connection or another, however, there may be an advantage from a system noise perspective. Vss is probably the most typical choice. Connecting directly to Vcc or Vss is not recommended since a direction register getting accidentally set to an output could create a shorted output.

The second method sets the pin to an output. It does not really matter whether the pin level is set high or low, however, selecting the pin as an output and low connects the pin internally to the ground plane. This may help with overall system noise concerns. One disadvantage the method of setting the port pin to an output is that the configuration of the port must be done via software control. While the MCU is held in Reset and until the direction register is set for output the pin will be a gloating input. However, if the extra current can be tolerated during this time this does eliminate the pull-up resistors required in previous method.

A variation on leaving the pins as inputs uses the internal pull-up resistors of the MCU. This has the same limitation as setting the pins to outputs (requires the program to set the port up) but it does limit the effect of accidental pin shorts to ground, adjacent pins or Vcc since the device will not be driving the pin.

6. Clock Circuits

6.1. Reset Conditions

- On Power-Up and after a Reset the MCU will be running on the Slow Speed On-Chip Oscillator with the CPU clock (Bclk) divided by 8. This results in a core clock speed of approximately 125 kHz/8 or 15 kHz. It is, therefore, important to shift to a higher speed clock early in the program if startup time is important
- The High Speed On-Chip Oscillator trim register will be loaded with the calibrated value to provide a 40 MHz clock as part of a Reset.

6.2. Clock Protection

- Changes to the clock mode registers are protected by the prc0 bit in the PRC register. It is recommended that the protection bit only be set to disable writing to these registers except when purposely modifying those registers

6.3. Low Power States (Wait and Stop Mode)

- Please see app notes
  - Using Low Power Wait Mode for R8C/2X
  - Using Stop Mode on R8C/2x

  For more discussion on these special power modes

7. Protection Register

The protection register allows protecting some registers that can be important in an application. These registers are:

Clock Mode Registers
Processor Mode Register
Flash Control Registers
Port 0 Direction Register
Voltage Detect Control Registers

All the registers can be left unprotected by modifying the associated protection register bit except for the Port 0 Direction Register. The first write to the memory space after unprotecting this register (setting the protect bit to 1) automatically protects this register (sets the bit back to 0). Due to this operation it is recommended that interrupts be disabled and any DMA operations be prohibited during these instructions. A typical sequence is shown below

```
asm (“fclr I”); //disable global interrupts
prc2 = 1;              // allow writing PD0 direction register
pd0 = 0xff ;           // set all of port 0 to outputs
prc2 = 0;              // not actually required but good practice
asm (“fset I”)  ;      // enable global interrupts
```

8. Interrupts

The R8C has individual interrupts for nearly all peripherals. The interrupts can have individual priorities and separate vector locations for the interrupt service routines. The M16C family, including the R8C, has two interrupt vector tables. One table has fixed addresses which we will refer to as the fixed interrupt table. The other table can be relocated by changing the value of the INTB core register, which is referred to as the relocatable vector table**.**

8.1. Fixed vector table

The fixed vector table contains the vector addresses for the system type non-maskable interrupts. These interrupts include:

Undefined instruction
Overflow
Break
Address Match
Single Step
WDT, Voltage Monitor and Oscillation Stop Detect
Address Break
A reserved vector
Reset

The Reset vector is located from 0FFFC to 0FFFD. The vector table works down in memory (lower addresses) from there with each vector location occupying four bytes. The fixed vector table is also unique since part of this area has some other special features. Since each vector address is actually only 3 bytes (24 bit addressing) the upper byte of many of the vectors is used for special functions. It is worth noting since changing the vector address in for these interrupts can change these values.

8.2. Shared functions in fixed vector table

8.2.1. Reset Vector - The upper byte (0ffff) of the Reset vector contains the following special functions:

- Controlling whether WDT is on by default out of reset
- Controlling if WDT Count source protect (independent clock) is used for WDT
- Controlling ROM code protect, which allows disabling parallel programmers from accessing the flash

This memory location is referred to as the Option Function Select Register (OSFR) and is discussed in section 19.3.2 of the hardware manual

8.2.2. All other fixed vectors except Break Vector
(Addresses 0FFDF, 0FFE3,OFFEB,OFFEF,OFFF3,0FFF7,OFFFB)
These 7 bytes make up the serial ID code which can be used to protect the device from being erased or read using the serial flash interface. When communicating with the device using the serial program interface the serial interface must first be unlocked by sending down a ID code that matches the one stored in these locations. Since these locations are not

normally used, if an ID code is not intentionally written to these locations the ID code will typically be all 00's or all FF's

8.2.3. Modifying the Shared Features in Vector Table
The High Performance Embedded Workshop IDE has provisions to allow setting both the OSFR register and ID code values. These options can be found by going to the Build >> Renesas M16C Standard Toolchain then LMC tabs. The options will be under the "Code" category.

8.3. The Relocatable vector table

The relocatable vector table contains the vector address for the peripheral interrupts and software interrupts. The start location (lowest address) of the vector table is pointed to by the value in the INTB core register. A few other features of the peripheral vectors:

- Most peripherals have individual interrupts or individual interrupts for each major function (e.g. UART Rx and UART Tx). This simplifies ISR routine writing
- The peripheral interrupts can be disabled by clearing the interrupt enable bit in the core, (I bit). This will globally disable all maskable interrupts
- Each peripheral can also be individually enabled or disable by controlling the interrupt priority level (IPL) in the associated interrupt control register for that peripheral. If the IPL can be set from 0 – 7. Setting the IPL to 0 will disable the interrupt from that device and setting the IPL to 7 will give it the highest priority

8.4. Interrupt Sequence

When a peripheral interrupt occurs the device sets the Interrupt Request (IR) bit in the associated interrupt control register for that peripheral. This will occur whether the interrupts for that device are enabled or not. If the device interrupts are enabled, global interrupts are enabled and the IPL of the device is higher than the current IPL of the core (which is 0 unless another interrupt is in the process of being serviced) the following interrupt sequence is followed:

I <u>Interrupt Sequence</u>

<u>Request Generation</u>

1. The peripheral device condition generates an interrupt
2. The Request (IR) bit in the interrupt control register for that device is set (regardless of the status of the IPL or I flag)
3. If global interrupts are enabled and IPL of the device is higher than the current MCU IPL (stored in the flag register and set to 0 unless there is an interrupt currently being serviced or set to a value by software) the interrupt is accepted

<u>Interrupt Service Response</u>

4. The current status of the flag register is stored in hidden register in core
5. I flag is set

6. The CPU gets interrupt information (interrupt number and interrupt request level) by reading address 00000h.
7. The IR bit for the corresponding interrupt is set to 0 (interrupt not requested))
8. The FLG register is saved to a temporary register(1) in the CPU immediately before entering the interrupt sequence.
9. The I, D and U flags in the FLG register are set as follows:
    - The I flag is set to 0 (interrupts disabled).
    - The D flag is set to 0 (single-step interrupt disabled).
    - The U flag is set to 0 (ISP selected).
      However, the U flag does not change state if an INT instruction for software interrupt number 32 to 63 is executed.
10. The CPU's internal temporary register is saved to the stack.
11. The PC is saved to the stack.
12. The interrupt priority level of the acknowledged interrupt is set in the IPL.
13. The starting address of the interrupt routine set in the interrupt vector is stored in the PC.

8.5. Using R8C Interrupts

8.5.1. Clearing the Request Bit and Multiple Interrupts

- Note that the IR bit for the interrupt is cleared prior to entering the ISR. If a second interrupt cause for the same peripheral happens while in the ISR that occurrence will be latched and the MCU will immediately re-enter the ISR it just finished. Some common issues that this causes:
- Switches bouncing will be recorded as two or more "pushes"
- A timer can actually "lock" the code by continually looping the ISR if the ISR is longer than the delay time of the timer

8.5.2. User Stack and Interrupt Stack Pointers

Note that the status of the U flag (user stack flag) is stored prior to entering the interrupt and restored when the interrupt ends. The Interrupt stack is always used to store the PC and flag information for the interrupt. Therefore there must always be a valid interrupt stack space and stack pointer. However, a user stack is not required if the U flag is never set to 1 by the program, the MCU hardware does not set the flag to 1, it only restores it to the value prior to the ISR (The U flag is cleared after reset so the default stack is the interrupt stack)

8.5.3. Code and ISR's

- Make sure all ISR's are declared using the #pragma INTERRUPT {function declaration}. So the compiler will use a REIT (return from interrupt) instruction rather than RTS (return from subroutine) instruction
- If the interrupt declaration is made using the following notation
  #pragma INTERRUPT {function declaration} /B
  The compiler will automatically use bank switching rather than stack pushes to save the contents of the core register, improving ISR response
- If the interrupt declaration is made using the following notation

#pragma INTERRUPT {function declaration} /E

The compiler will automatically re-enable interrupts during the ISR to allow nested interrupts

- You cannot use /B and /E on the same interrupt declaration (you will not get an error it just won't generate the code for those switches)
- The IR bit can be cleared by software but cannot be set by software.

8.5.4. Special Interrupt Usage Notes

- The operation of Timer RD, Synchronous Serial Unit (SSU) and I2C peripherals is slightly different than described since these vectors have multiple sources. Please refer to the manual for the exact operation of those IR bits

- When using the Key Input Interrupts, if any enabled Key Input is held in the low state it will mask any further interrupts from other keys.